7N-31-TM

The Need For Vendor Source Code at NAS

Russell Carter

Report RND-007 June 1994

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

The Need For Vendor Source Code at NAS

Russell Carter

Report RND-007 June 1994

NAS Systems Development Branch
NAS Systems Division
NASA Ames Research Center
Mail Stop 258-5
Moffett Field, CA 94035-1000

Abstract

The Numerical Aerodynamic Simulation (NAS) Facility has a long standing practice of maintaining buildable source code for installed hardware. There are two reasons for this: NAS's designated pathfinding role, and the need to maintain a smoothly running operational capacity given the widely diversified nature of the vendor installations. NAS has a need to maintain support capabilities when vendors are not able; diagnose and remedy hardware or software problems where applicable; and to support ongoing system software development activities whether or not the relevant vendors feel support is justified. This note provides an informal history of these activities at NAS, and brings together the general principles that drive the requirement that systems integrated into the NAS environment run binaries built from source code, *onsite*.

1.0 Introduction

The Numerical Aerodynamic Simulation (NAS) Program at Ames Research Center is a large scale integrated computer center featuring vector supercomputers, highly parallel systems, high performance graphics workstations and appropriately large storage servers. These systems run various flavors of Unix, and are all interconnected via one or more high performance LAN. NAS supports over fifteen major systems and 350 systems overall. Each subsystem is characterized by production use of multiple vendor solutions, which must coexist peacefully in the open environment. The responsibility of the NAS Development and Support Branches (RND and RNS) is the continued smooth functioning of this very complex environment.

A major goal of the NAS division is pathfinding in the area of high performance system integration. Achieving this goal requires the flexibility to install the highest performing systems, regardless of vendor. Thus, the NAS pathfinding goal is the source of much of the diversity, and accompanying challenges, inherent in the NAS operational environment.

It is critical for the tools and techniques used to maintain the effectiveness of the NAS environment that the operating system and layered software environments be run from source code built onsite. The list of vendors includes Wellfleet, Cray Research, Thinking Machines, Silicon Graphics, Sun, Convex, BSDI, StorageTek, Ultra, and Intel Scientific Computers. Each of these vendors supplies system source code, and in most cases the running operating system is built onsite. NAS personnel frequently respond to questions about the origin and necessity of the buildable source code requirement, particularly from vendors unfamiliar with or new to the NAS environment. This document provides a sampling of our experiences, pros and cons, with operating system issues that pertain to the NAS requirement that "we build what we run," and thereby should provide insight into the motivation behind our rigid requirement that buildable operating system source code must accompany any system installed at our site.

The structure of this document is as follows. First, there is a description of the real-world vendor environment in which NAS operates. Next follows the three categories of use that we have for system source code:

- Documentation
- Software Development
- Bug identification, patch; resolution of vendor design flaws

Then follows a discussion of vendor data rights, and some of our procedures that we use to protect them. Finally, there is a summary. It is important to note that no proprietary code is ever transferred to unlicensed platforms at NAS—even for projects that require porting.

2.0 Practical Considerations

Many vendors sell a product and offer little or no support for integration into complex multivendor sites like NAS. The product may work fine under some circumstances but due to the complexity of software and configuration issues, it may not work as intended for NAS.

Reasons for the lack of support are varied but are usually one or more of the following:

- Hard problems!
  (Complex systems have complex problems!)
- Management issues:
  Magnitude of support task underestimated
  Poor internal communications
  Support staff not competent enough
  Insufficient engineering resources
- Firm having financial difficulties
- Conflicts with marketing/sales goals

In the last item, a firm may be overly sales-oriented, particularly when a new product is introduced. The vendor may not give sufficient priority to support, or a vendor may choose not to recognize an issue as a bug due to sales/marketing ramifications.

3.0 Experiences at NAS

The experiences fall roughly into the four different categories listed below. I have included typical examples. This is by no means comprehensive: the list I maintain for these examples has many more entries.

3.1 Documentation

Documentation on the types of systems installed at NAS is often out of date with the installed software, or even just wrong. Some documentation may be out of date due to running early releases or beta software. Others are out of date even though packaged in an official release. Fundamental tasks such as adding a device driver, say for a HiPPI attached RAID array are much more difficult without access to source, since the definitive documentation on what kernel calls (for instance) are available is the source code itself.

- On Cray Research Systems, source code was used to fix incorrect documentation. The actual system scheduling parameters turned out to be entirely different from claims on the manual page. Similarly, incomplete documentation was filled in by using source code. Exactly how the scheduler determines an "interactive" process was discovered this way.
- The Portable Batch System's resource monitor needs to be able to extract information from the system kernel concerning global resource availability and utilization and about per-session resource utilization. PBS's machine oriented miniserver module has similar needs to control session limits. Generally, documentation of the necessary kernel interfaces is sketchy enough that reference to code is required to find out how they work. In the case of the Paragon, the documentation is completely lacking. Fortunately, persons working on the Portable Batch System can refer to the source code to learn what is needed.
- Wellfleet source code provided the only accurate documentation on the workings of the supplied routers. This knowledge was necessary to develop effective network monitors.
- Rebuilding everything from vendor-supplied source guarantees that you have the correct source. Problems with Cygnus Support distributions were uncovered by building from source onsite.
- The *top* program, widely used for system monitoring, relies on kernel tables. Often the tables are not documented.

3.2 Security

Due to the wide variety of operating systems and software applications installed at NAS, there are a large number of security alerts that must be evaluated by NAS personnel. Many times, unofficial security alerts are posted by users who have had systems compromised long before official security alerts and vendor patches are available. During such time, it is essential that source code be available to NAS personnel in order to evaluate the level of risk and/or to modify code in the event that the security bug is of a critical nature. Also, there are times when security needs to be strengthened within operating systems and applications in order to meet local security policies. Such issues also indicate the need for source code when vendor supplied security patches involve portions of software that previously have been modified to meet NAS needs.

- Sun and SGI source code allowed for the modification of *rpc.mountd* which increased the level of NAS network security by limiting NFS mounting of NAS machines from only those machines within the *nas.nasa.gov* domain.

3.3 Bug identification, patching; resolution of vendor design flaws

There are numerous examples of site-critical bugs fixed using source code before the vendor could (or would) respond to bug report.

- Incorrect padding of tape blocks in a tape driver ruined backups. Botched system scheduling parameters required fixing.
- The Proteon Ring evaluation required driver debugging using UNICOS source code.
- Early SGI releases included a bug that prevented telnet connections to the Cray-2. Source code was required to fix the bug.
- NAStore, the NAS mass storage system, used an Informix data base which resulted in significant waste of time and personnel. Source code was not obtained because of the high cost. It turned out that the software had critical bugs relating to its *commit* function. Informix was unwilling to fix them. That cost many man-months to implement workaround and recovery procedures, and more to do what should have done in the first place -- reimplement NAStore only around packages for which source code is available, in this case a b-tree package.
- The Morris Virus *sendmail* bug was fixed using source code for all systems except SUN, within two days. Without source code to SUN *sendmail*, the fix for SUN systems took six months.
- Resolution of vendor-vendor incompatibilities is facilitated through the use of diagnostic traps implemented in the operating systems, as was the case with Cray and NSC.
- iPSC/860 remote host software ported to SGI systems. Without this added functionality, the iPSC/860 could not have supported the NAS workload. NAS personnel successfully carried out the port.
- iPSC/860 cube limit increased to more than ten. The artificial cube limit severely impacted the usability of the iPSC/860 to support multidisciplinary codes.
- Routing protocol bugs in the Wellfleet routers were fixed in order to support the Routing Information Protocol package. Workarounds for other problems were implemented through this manner as well.
- Amdahl File system problems. After a catastrophic crash of the Amdahl hosted mass storage system, NAS personnel diagnosed an Amdahl generated fatal design error. Access to source code allowed the implementation of file system repair tools that Amdahl had never bothered to produce, with a partial recovery of the lost data.

3.4 Software Development

The nature of the systems installed at NAS requires ability to add new features and functionality that frequently the vendor will not agree to do for us in a timely fashion, if at all. NAS acquires many technologies before they are "proven" or "mature", and often, making such a technology work in production implies that we will perform development support on the technology. Without source code, the vast majority of this work would not be possible and NAS would be subjected to the sometimes unsuitable agendas of vendors for support and further development. The following examples illustrate the varieties of NAS development needs that are met using source code.

- The Network Queuing System (NQS) was developed using source code for targeted systems.
- The Portable Batch System's resource monitor needs to be able to extract information from the system kernel concerning global resource availability and utilization and about per-session resource utilization. PBS's machine oriented miniserver module has similar needs to control session limits. Generally, documentation of the necessary kernel interfaces is sketchy enough that reference to code is required to find out how they work.
- Amdahl UTS source code enabled the implementation of NAStore and port TCP/IP when it wasn't available from Amdahl.
- ConvexOS source code allowed us to port NAStore and enhance the OS. Joint development with Convex resulted in an improvement in file system performance from 20 MB/s to 150 MB/s.
- Wellfleet routers source code allowed the implementation of router discovery. This is now a distributed product by Wellfleet.
- Source for Unicos and Cray NQS allowed the implementation of the Session Reservable File System, an enhanced resource management function. Disk quotas and an Ultra driver (mandatory in the NAS environment) were also implemented.
- UNICOS source code was need to develop tools to modify the kernel mount table and develop the *top*, and *mu* (memory usage) commands. Minor hooks were added for new user-level services such as real-time and cpu-time gid limits.
- Non-intrusive, low-level data collection requires modification to the operating system. Two successful projects that required access to system source code are the iPSC/860 Concurrent File System monitoring and Van Voorst's message sizes monitoring project.
- The Map library on the iPSC/860 to support multidisciplinary applications is a modified version of an Intel message passing library, successfully modified with no performance degradation. It allows application programmers access enhanced message passing functionality.
- For the p2d2 project and MPK project, source is required on targeted platforms for the libraries that support parallel processing, e.g.,

message passing, collective operations, process creation, termination, locks, events, and critical sections.

4.0 Vendor Data Rights

NAS recognizes the sensitivity of vendor source code, and strives to protect the source code that we are given access to, while at the same time maintaining efficient development practices. The procedures in place have allowed us to maintain the NAS's functionality while in no instance causing damage through disclosure to unapproved parties, in the ten year history of the program. For instance, NAS personnel do not take licensed software from Vendor A and port it to Vendor B's product, be it hardware or software products. Also, Vendor A does not have access to Vendor B source code. The source distributions are kept in separate areas.

It has been the custom at NAS to negotiate the terms and conditions for buildable source code access at the initial acquisition stage. This has been found to be the only practical way of dealing with numerous and often competing interests surrounding the issue.

The policy at NAS is to support local modifications to source code, if the modifications are not adopted by the vendor. On the other hand, NAS modifications are willingly transferred to the vendor. In several instances, these are now part of revenue generating packages offered to other customers by the vendor.

5.0 Summary

In an ideal world, hardware works flawlessly, software works flawlessly, kernel interfaces are documented completely and flawlessly, documentation is in sync with binary distributions, and vendors are strongly motivated to meet NAS mission critical needs in a timely manner. None of these hold in practice. Hence the requirement that NAS run system software built onsite from source code. NAS access to buildable source code has demonstrable benefits for the vendor, and historically has been a basis for continued progress and improvement of the supplied systems, which has lead to increased markets and revenue for vendors.

6.0 Acknowledgments

This note represents my distillation of the collective wisdom of the NAS Systems Development (RND) and Computational Services (RNS) Branch personnel, including supporting contractors. Without the help of the many persons who have worked on the complex systems problems historically encountered at the NAS, this information would not have been documented. Indeed, much of the information in this note has not previously been formally documented from the perspective of the source code requirements. Contributors of comments, examples and/or text include (in no particular order): Bruce Blaylock, John Lekashman, Robert Ciotti and Dave Tweten of NASA RND; Bill Kramer, and Toby Harness of NASA RNS; Jonathan Hahn of UNETIX; Parkson Wong, Bill Nitzberg, Sam Fineberg, Bernard Traversat, Dave McNab, Eric Townsend, Keith Thompson, Alfred Nothaft, Doreen Cheng, Jeff Becker, and David Barkai of Computer Sciences Corporation; and D. V. Henkel-Wallace of Cygnus Support.

RND TECHNICAL REPORT

Title: The Need for Vendor Source Code at NAS

Author(s): Russell Carter

Reviewers:

"I have carefully and thoroughly reviewed this technical report. I have worked with the author(s) to ensure clarity of presentation and technical accuracy. I take personal responsibility for the quality of this document."

Signed: David Barkai

Name: DAVID BARKAI

Signed: [illegible]

Name: William TC Kramer

Branch Chief:

Approved: [illegible]

Date & TR Number:

June 94 RND-94-007